

# The LEOS Interpolation Package

*F. N. Fritsch*

**March 12, 2003**

**U.S. Department of Energy**

Lawrence  
Livermore  
National  
Laboratory

This document was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

# The LEOS Interpolation Package

Third Edition

Frederick N. Fritsch

Physics and Advanced Technology Directorate  
H Division

12 March 2003

(This page deliberately left blank.)

# The LEOS Interpolation Package

## Contents

Contents .....	iii
Introduction.....	1
1. Bivariate functional forms .....	1
1.1. Variable transformation .....	1
1.2. The bilinear form .....	1
1.3. The bilogarithmic form.....	2
1.4. The bicubic form.....	2
1.5. The bicubic Hermite form.....	2
1.6. The biquadratic form .....	3
2. Univariate functional forms .....	3
2.1. The need for compatible univariate interpolation.....	3
2.2. Linear interpolation .....	3
2.3. Cubic interpolation .....	4
2.4. Cubic Hermite interpolation .....	4
3. Coefficient setup .....	4
3.1. Bilinear.....	4
3.2. Bicubic .....	5
3.3. Modification for two-phase data.....	7
3.4. Bicubic Hermite .....	8
3.5. Monotone bicubic Hermite (bimond) .....	10
3.6. Biquadratic.....	11
3.7. Linear.....	11
3.8. Cubic.....	12
3.9. Cubic Hermite.....	12
3.10. Monotone cubic Hermite (monder) .....	13
4. Forward evaluation .....	13
4.1. Table lookup .....	13
4.2. Evaluation .....	14
4.3. Extrapolation.....	14
5. Inverse evaluation .....	15
5.1. Table lookup .....	15
5.2. Direct inversion (bilinear).....	16
5.3. Inverse iteration (bicubic).....	16
5.4. Inverse iteration (biherm) .....	16
5.5. Inverse iteration (biquadratic).....	16
5.6. Extrapolation.....	16
6. The Tcalc option .....	17
6.1. The Tcalc mesh.....	17
6.2. Tcalc setup .....	17
6.3. Tcalc evaluation.....	18
6.4. Tcalc extrapolation .....	18
7. Package organization .....	18
7.1. The interpolation utilities.....	18
7.2. Interp1D .....	19
7.3. Interp2D .....	20

8. Possible enhancements .....	22
8.1. Lookup improvements .....	22
8.2. Newton iteration .....	22
References.....	23
Acknowledgements.....	23

## Introduction

This report describes the interpolation package in the Livermore Equation of State (LEOS) system. It is an updated and expanded version of report [1], which described the status of the package as of May 1998, and of [2], which described its status as of the August 2001 release of the LEOS access library, and of [3], which described its status as of library version 7.02, released April 2002. This corresponds to library version 7.11, released March 2003. The main change since [3] has been the addition of the monotone bicubic Hermite (bimond) interpolation method.

Throughout this report we assume that data has been given for some function  $f(\rho, T)$  on a rectangular mesh  $\rho = \rho_0, \rho_1, \dots, \rho_{nr-1}$ ;  $T = T_0, T_1, \dots, T_{nt-1}$ . Subscripting is from zero to be consistent with the C code. (Although we use this notation throughout, there is nothing in the package that assumes that the independent variables are actually density and temperature.)

The data values are  $f_{ij} = f(\rho_j, T_i)$ . (This subscript order is historical and reflects the notation used in the program.) There are  $nr \times nt$  data values,  $(nr-1) \times (nt-1)$  mesh rectangles (boxes). In the C code, the data array is one-dimensional, with  $data[i * (nr-1) + j] = f(\rho_j, T_i)$ . In the case of the few univariate functions supported by LEOS, the  $T$  variable is omitted, as well as the associated index on the data array:  $data[j] = f(\rho_j)$ .

## 1. Bivariate functional forms

### 1.1. Variable transformation

For the standard bivariate forms, the following variable transformations are used to simplify formulas and enhance numerical stability:

$$x = x(\rho) = (\rho - \rho_j) / \Delta\rho_j, \quad \Delta\rho_j = \rho_{j+1} - \rho_j; \quad (1.1)$$

$$y = y(T) = (T - T_i) / \Delta T_i, \quad \Delta T_i = T_{i+1} - T_i. \quad (1.2)$$

Note that these transformations have the property

$$x(\rho_j)=0, \quad x(\rho_{j+1})=1, \quad y(T_i)=0, \quad y(T_{i+1})=1, \quad (1.3)$$

so that the rectangle  $R_{ij}=[\rho_j, \rho_{j+1}] \times [T_i, T_{i+1}]$  is mapped onto the unit square  $U=[0,1] \times [0,1]$ .

These variable transformations are not performed in the biquadratic case (see Section 1.6), in order to produce an interpolant as close as possible to its equivalent in the old EOS4 package [7].

### 1.2. The bilinear form

The bilinear functional form on the mesh rectangle  $R_{ij}$  is:

$$f(\rho, T) = l(x, y) = a_0 + a_1 x + a_2 y + a_3 xy. \quad (1.4)$$

## 1. Bivariate functional forms

### 1.3. The bilogarithmic form

The bilogarithmic functional form (abbreviated “bilog”) is

$$\log f(\log \rho, \log T) = l(x,y) = a_0 + a_1x + a_2y + a_3xy. \quad (1.4bl)$$

This was recently introduced as an option, primarily for opacity data. If this option is chosen, logarithms are taken of all variables before setup, and the bilinear interpolation coefficients are computed for the resulting transformed data. Since the user will be supplying  $(\rho, T)$  and expecting  $f$  back, the necessary transformations are done before and after the evaluation is performed.

### 1.4. The bicubic form

The LEOS bicubic functional form on the mesh rectangle  $R_{ij}$  is:

$$\begin{aligned} f(\rho, T) = c(x,y) = & a_0 + a_1x + a_2y + a_3xy + \\ & a_4x^2 + a_5x^2y + a_6x^3 + a_7x^3y + \\ & a_8y^2 + a_9xy^2 + a_{10}y^3 + a_{11}xy^3, \end{aligned} \quad (1.5)$$

where the four highest-order terms  $(x^2y^2, x^3y^2, x^2y^3, x^3y^3)$  have been omitted from the general bicubic polynomial to reduce storage space and evaluation time. Because the first four terms in (1.5) are the same as in (1.4), we note that a bilinear function can be viewed as a bicubic with its last eight coefficients equal to zero.

### 1.5. The bicubic Hermite form

The bicubic Hermite functional form (abbreviated “biherm”) on the mesh rectangle  $R_{ij}$  is:

$$\begin{aligned} f(\rho, T) = b(x,y) = & a_0 h_0(x)h_0(y) + a_1 h_1(x)h_0(y) + a_2 h_2(x)h_0(y) + a_3 h_3(x)h_0(y) + \\ & a_4 h_0(x)h_1(y) + a_5 h_1(x)h_1(y) + a_6 h_2(x)h_1(y) + a_7 h_3(x)h_1(y) + \\ & a_8 h_0(x)h_2(y) + a_9 h_1(x)h_2(y) + a_{10}h_2(x)h_2(y) + a_{11}h_3(x)h_2(y) + \\ & a_{12}h_0(x)h_3(y) + a_{13}h_1(x)h_3(y) + a_{14}h_2(x)h_3(y) + a_{15}h_3(x)h_3(y). \end{aligned} \quad (1.6)$$

For improved numerical stability, we have performed a change of basis on the bicubic form. Note that this is not equivalent to (1.5), because it contains the full 16 terms required for a general bicubic function. Note also that a bilinear function is *not* a special case.

The (univariate) *cubic Hermite basis functions* that appear in (1.6) are defined by relations:

$$h_0(0) = 1, \quad h_0(1) = 0, \quad h_0'(0) = 0, \quad h_0'(1) = 0; \quad (1.7a)$$

$$h_1(0) = 0, \quad h_1(1) = 1, \quad h_1'(0) = 0, \quad h_1'(1) = 0; \quad (1.7b)$$

$$h_2(0) = 0, \quad h_2(1) = 0, \quad h_2'(0) = 1, \quad h_2'(1) = 0; \quad (1.7c)$$

$$h_3(0) = 0, \quad h_3(1) = 0, \quad h_3'(0) = 0, \quad h_3'(1) = 1. \quad (1.7d)$$

These lead to the following formulas for the basis functions:

## The LEOS Interpolation Package

$$h_0(t) = 1 - 3t^2 + 2t^3 = h_1(1-t) = u^2(u + 3t); \quad (1.8a)$$

$$h_1(t) = 3t^2 - 2t^3 = t^2(t + 3u); \quad (1.8b)$$

$$h_2(t) = t - 2t^2 + t^3 = -h_3(1-t) = u^2t; \quad (1.8c)$$

$$h_3(t) = -t^2 + t^3 = -t^2u, \quad (1.8d)$$

where we have set  $u = 1 - t$ .

### 1.6. The biquadratic form

The biquadratic functional form on the mesh rectangle  $R_{ij}$  is:

$$f(\rho, T) = q(\rho, T) = a_0 + a_1\rho + a_2T + a_3\rho^2 + a_4T^2 + a_5\rho T + a_6\rho^2T + a_7\rho T^2 + a_8\rho^2T^2. \quad (1.9)$$

Recall that there is no transformation to the normalized variables  $(x, y)$  in this case.

The biquadratic form is not recommended as a general bivariate interpolator. It is included in the LEOS interpolation package only as a means to provide an interpolant to the old EOP data that is consistent with EOS4 [7].

## 2. Univariate functional forms

### 2.1. The need for compatible univariate interpolation

In cases where the univariate “cold curves” are provided by LEOS, it is expected that

$$F_t = F_c + F_e + F_i, \quad (2.1)$$

where the second letters stand for total (t), cold (c), electronic (e) and ionic (i).  $F_c$  is univariate (a function of  $\rho$  only), and the others are bivariate. (Here  $F = E$  or  $P$ .) The univariate interpolant needs to be compatible with its bivariate interpolant of the same type (bilinear, bicubic, or biherm), in the sense that if the data for these four functions satisfy (2.1), then so will the interpolants (to as close to machine precision as possible).

The univariate interpolator in the 1998 version was a cubic spline, which is not compatible with either the bilinear or bicubic form. The current package contains compatible linear, cubic, and cubic Hermite interpolators. Since the EOP data is available only for  $F_t$ , there is no univariate quadratic interpolator.

### 2.2. Linear interpolation

The linear functional form on the mesh interval  $[\rho_j, \rho_{j+1}]$  is:

$$f(\rho) = l(x) = a_0 + a_1x, \quad (2.2)$$

where, unlike in (1.1), we do not divide by the interval length:

$$x = x(\rho) = \rho - \rho_j. \quad (2.3)$$

### 3. Coefficient setup

#### 2.3. Cubic interpolation

The cubic functional form on the mesh interval  $[\rho_j, \rho_{j+1}]$  is:

$$f(\rho) = c(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3, \quad (2.4)$$

where  $x$  is as in (2.3).

#### 2.4. Cubic Hermite interpolation

The cubic Hermite functional form on the mesh interval  $[\rho_j, \rho_{j+1}]$  is:

$$f(\rho) = c(x) = a_0 h_0(x) + a_1 h_1(x) + a_2 h_2(x) + a_3 h_3(x), \quad (2.5)$$

where  $x$  is in (1.1), and the  $h_i(x)$  are as in (1.8).

## 3. Coefficient setup

#### 3.1. Bilinear

The interpolation conditions on mesh rectangle  $R_{ij}$  are:

$$f_{mn} = f(\rho_n, T_m), \quad m = i, i+1, \quad n = j, j+1. \quad (3.1)$$

There are four coefficients in (1.4) and four conditions in (3.1). Using (1.3), we can write down the coefficients immediately from (3.1).

$$f_{ij} = f(\rho_j, T_i) = l(0,0) = a_0; \quad (3.2a)$$

$$f_{i,j+1} = f(\rho_{j+1}, T_i) = l(1,0) = a_0 + a_1; \quad (3.2b)$$

$$f_{i+1,j} = f(\rho_j, T_{i+1}) = l(0,1) = a_0 + a_2; \quad (3.2c)$$

$$f_{i+1,j+1} = f(\rho_{j+1}, T_{i+1}) = l(1,1) = a_0 + a_1 + a_2 + a_3. \quad (3.2d)$$

(3.2a) gives us  $a_0$  directly:

$$a_0 = f_{ij}. \quad (3.3a)$$

From (3.2b) and (3.3a) we have

$$a_1 = f_{i,j+1} - f_{ij}. \quad (3.3b)$$

Similarly, (3.2c) and (3.3a) yield

$$a_2 = f_{i+1,j} - f_{ij}. \quad (3.3c)$$

Finally, (3.2d) gives us

$$a_3 = f_{i+1,j+1} - (a_0 + a_1 + a_2). \quad (3.3d)$$

We observe that, by construction, the bilinear form will be continuous across the box boundaries. However, derivatives will have jump discontinuities there.

## The LEOS Interpolation Package

The interpolation coefficients are laid out in blocks of four in memory, with the coefficients for mesh rectangle  $R_{ij}$  starting at location  $(i * (nr-1) + j) * 4$  in the coefficient array.

### 3.2. Bicubic

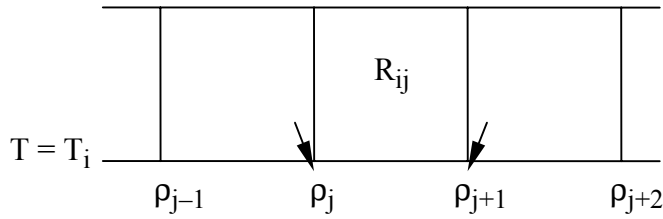
The same four interpolation conditions (3.1) apply to the bicubic form on mesh rectangle  $R_{ij}$ , but there are twelve coefficients in (1.5), so we must find eight additional equations. If we had the values of the first partial derivatives of  $f$  at the data points, we could supplement (3.1) with the eight derivative interpolation conditions:

$$D_{\rho}f_{mn} = \partial f(\rho_n, T_m) / \partial \rho, \quad m = i, i+1, \quad n = j, j+1. \quad (3.4a)$$

$$D_Tf_{mn} = \partial f(\rho_n, T_m) / \partial T, \quad m = i, i+1, \quad n = j, j+1. \quad (3.4b)$$

where  $D_{\rho}f_{mn}$  is the  $\rho$ -derivative of  $f$  at the  $mn$  data point, and similarly for  $D_Tf_{mn}$ . To approximate the needed derivatives, we bring in information from neighboring points.

First observe that the four points  $(\rho, f(\rho, T_i))$ ,  $\rho = \rho_{j-1}, \rho_j, \rho_{j+1}, \rho_{j+2}$  determine a (univariate) cubic in  $\rho$ . The present LEOS interpolator evaluates the derivative of this cubic at  $\rho_j$  and  $\rho_{j+1}$  to provide estimates of  $D_{\rho}f_{ij}$  and  $D_{\rho}f_{i,j+1}$  in (3.4a).



Requiring the derivatives of bicubic (1.5) to match these two values gives two additional equations. Applying this procedure at  $T=T_{i+1}$  gives two more. The other four equations are determined similarly, by reversing the roles of  $T$  and  $\rho$  and using (3.4b) instead of (3.4a).

Differentiating (1.5), and taking (1.1) and (1.2) into account, gives the partial derivatives required for (3.4a) and (3.4b) in  $R_{ij}$ :

$$\partial f(\rho, T) / \partial \rho = \partial c(x, y) / \partial x / \Delta \rho_j = [a_1 + a_3 y + 2a_4 x + 2a_5 xy + 3a_6 x^2 + 3a_7 x^2 y + a_9 y^2 + a_{11} y^3] / \Delta \rho_j. \quad (3.5a)$$

$$\partial f(\rho, T) / \partial T = \partial c(x, y) / \partial y / \Delta T_i = [a_2 + a_3 x + a_5 x^2 + a_7 x^3 + 2a_8 y + 2a_9 xy + 3a_{10} y^2 + 3a_{11} xy^2] / \Delta T_i. \quad (3.5b)$$

Note that properties (1.3) greatly simplify the matrix setup. In fact, unlike the procedure used in the 1998 version of the package, the matrix is now constant, independent of the data. It is set up for interpolation on the unit square with the first four rows of the matrix containing the data interpolation conditions,  $f(\rho[i], t[i]) = rhs[i]$ :

### 3. Coefficient setup

```
rho[ 0] = 0;  t[ 0] = 0;
rho[ 1] = 0;  t[ 1] = 1;
rho[ 2] = 1;  t[ 2] = 0;
rho[ 3] = 1;  t[ 3] = 1.
```

Note that the internal variables `rho` and `t` have replaced the `x` and `y` used in (1.5). Rows 4–7 contain  $\rho$ -derivative interpolation conditions,  $\partial f(\text{rho}[i], t[i]) / \partial \rho = \text{rhs}[i]$ :

```
rho[ 4] = 0;  t[ 4] = 0;
rho[ 5] = 1;  t[ 5] = 0;
rho[ 6] = 0;  t[ 6] = 1;
rho[ 7] = 1;  t[ 7] = 1.
```

Rows 8–11 contain  $T$ -derivative interpolation conditions,  $\partial f(\text{rho}[i], t[i]) / \partial T = \text{rhs}[i]$ :

```
rho[ 8] = 0;  t[ 8] = 0;
rho[ 9] = 0;  t[ 9] = 1;
rho[10] = 1;  t[10] = 0;
rho[11] = 1;  t[11] = 1.
```

Note that the  $\partial f / \partial \rho$  conditions are not in the same order as the others. This is because it is natural to generate  $\rho$ -derivative estimates with  $T$  constant,  $T$ -derivative estimates with  $\rho$  constant.

The resulting constant matrix is (in C notation):

```
int imat[] = { 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \
               1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, \
               1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, \
               1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, \
               0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, \
               0, 1, 0, 0, 2, 0, 3, 0, 0, 0, 0, 0, \
               0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, \
               0, 1, 0, 1, 2, 2, 3, 3, 0, 1, 0, 1, \
               0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, \
               0, 0, 1, 0, 0, 0, 0, 0, 2, 0, 3, 0, \
               0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, \
               0, 0, 1, 1, 0, 1, 0, 1, 2, 2, 3, 3 }; (3.6)
```

The twelve coefficients in (1.5) are computed by solving a  $12 \times 12$  linear system, with the above matrix, using Gaussian elimination with partial pivoting. The procedure is to perform an LU factorization of the matrix exactly once. Then all of the linear systems solves reduce to a much more rapid back-substitution.

Because the necessary neighboring values needed to determine the univariate cubics are not available in the boundary boxes, the setup routine drops to quadratic in the normal direction, using only three-point estimates, in mesh rectangle  $R_{ij}$  if  $i = 0$  or  $nt-2$  or if  $j$

## The LEOS Interpolation Package

= 0 or nr-2. The boundary normal derivative is set to zero if this estimate has the opposite sign from the data slope.

Function values are generally continuous across the interior box boundaries, because the same function values and similar tangential derivative estimates are used on both sides of the boundary between two boxes. Although the derivatives are much smoother than with the bilinear form, they are not guaranteed to be continuous across box boundaries. In fact, they are generally *not* continuous in the normal direction. The tangential derivatives are nearly continuous, but this is not guaranteed. Because a different set of four data points is used to estimate  $D_{\rho}f_{ij}$  in box  $(i,j-1)$  and box  $(i,j)$ ,  $\partial f/\partial \rho$  may not even be continuous at  $\rho = \rho_j$  along the mesh lines. (Similar remarks apply to  $\partial f/\partial T$ .)

A version of the bicubic setup routine that uses the average of these two derivative estimates has been tested. Although the overall quality of the derivatives was better, the maximum derivative discontinuity was about the same as with the original method. This is presumably due to the fact that we are not using a complete 16-term bicubic here. It was decided not to change to these new estimates for such a marginal gain, because it would have changed the results for all existing users of the bicubic form. (Different derivative estimates lead to different interpolants.)

The interpolation coefficients are laid out in blocks of 12 in memory, with the coefficients for mesh rectangle  $R_{ij}$  starting at location  $(i*(nr-1)+j)*12$  in the coefficient array.

### 3.3. Modification for two-phase data

A modified version of the bicubic coefficient setup is used for two-phase data, because the standard bicubic behaves very badly at the edges of the two-phase region, where the data suddenly changes from being constant in  $\rho$  to changing very rapidly in this variable. The procedure is to drop to bilinear (a special case of bicubic) inside or at the boundary of the two-phase region.

The two-phase region is detected by the test (in C notation):

```
if ( (j > 0 && \
      isflat(data[ i      *nr + j-1], data[ i      *nr + j  ])) ||
    \
      isflat(data[ i      *nr + j  ], data[ i      *nr + j+1]) ||
    \
      (j < nr-2 && \
      isflat(data[ i      *nr + j+1], data[ i      *nr + j+2])) ||
    \
      (j > 0 && \
      isflat(data[(i+1)*nr + j-1], data[(i+1)*nr + j  ])) ||
    \
      isflat(data[(i+1)*nr + j  ], data[(i+1)*nr + j+1]) ||
    \
      )
```

### 3. Coefficient setup

```

      ( j < nr-2 && \
        isflat(data[(i+1)*nr + j+1], data[(i+1)*nr + j+2])) ) {
      goto Make_it_bilinear;
    }

```

(3.7)

Here the logical function `isflat` is defined by

$$\text{isflat}(a, b) = ( |a-b| / \max(|a|, |b|) \leq \text{FLATHRSH} ), \quad (3.8)$$

where the “flatness threshold” `FLATHRSH` is a code parameter that is currently equal to  $1.0e-7$ . To reduce the amount of extra testing, (3.7) is applied only in boxes  $R_{ij}$  which satisfy

$$T_i \leq 1.1 T_c, \quad \rho_j < 1.5 \rho_0,$$

where  $T_c$  is the “critical temperature” for the material and  $\rho_0$  is the “normal density”.

In order to reduce discontinuities between bicubic boxes and neighboring bilinear boxes, the derivative estimates at neighboring points are modified to match the linear slopes.

Note that the above procedure is intended primarily to handle the characteristics of two-phase pressure data. LEOS, however, currently applies this modified cubic setup to all of the two-phase functions: P2p, E2p, and S2p.

#### 3.4. Bicubic Hermite

The bicubic function  $b(x,y)$  on mesh box  $R_{ij}$ , after the transformations (1.1) and (1.2), is uniquely determined by the values of the four quantities

$$b, \partial b / \partial x, \partial b / \partial y, \partial^2 b / \partial x \partial y$$

at the corners of the box. By construction, if the same values of these four quantities are used in adjacent boxes, then these functions are continuous across the box boundaries. This requires using continuous derivative estimates for all partial derivatives that appear here.

A significant advantage of using the Hermite basis is that the interpolation coefficients can be established directly from the interpolation conditions without the need to solve any linear systems. For example, using the relations (1.7) in the functional form (1.6) immediately yields the interpolation conditions (3.1) and the following coefficients:

$$a_0 = b(0,0) = f(\rho_j, T_i) = f_{ij}; \quad (3.9a)$$

$$a_1 = b(1,0) = f(\rho_{j+1}, T_i) = f_{i,j+1}; \quad (3.9b)$$

$$a_4 = b(0,1) = f(\rho_j, T_{i+1}) = f_{i+1,j}; \quad (3.9c)$$

$$a_5 = b(1,1) = f(\rho_{j+1}, T_{i+1}) = f_{i+1,j+1}. \quad (3.9d)$$

Note that these four coefficients are in the upper left 4x4 corner of the coefficient matrix if equation (1.6) is rewritten in matrix notation:

$$b(x,y) = \mathbf{h}(y)^T \mathbf{A} \mathbf{h}(x), \quad (3.10a)$$

where

## The LEOS Interpolation Package

$$\mathbf{h}(t) = ( h_0(t), h_1(t), h_2(t), h_3(t) )^T, \quad (3.10b)$$

and

$$\mathbf{A} = \begin{pmatrix} a_0 & a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 & a_7 \\ a_8 & a_9 & a_{10} & a_{11} \\ a_{12} & a_{13} & a_{14} & a_{15} \end{pmatrix}. \quad (3.10c)$$

Differentiating (3.10a) with respect to  $x$  yields:

$$\partial b(x,y)/\partial x = \mathbf{h}(y)^T \mathbf{A} \mathbf{h}'(x), \quad (3.11)$$

Again applying (1.7) to the derivative interpolation conditions (3.4a) shows that the upper right corner of  $\mathbf{A}$  contains the  $\partial b/\partial x$ -values:

$$a_2 = \partial b(0,0)/\partial x = \Delta \rho_j \partial f(\rho_j, T_i)/\partial \rho = \Delta \rho_j D_{\rho} f_{ij}; \quad (3.12a)$$

$$a_3 = \partial b(1,0)/\partial x = \Delta \rho_j \partial f(\rho_{j+1}, T_i)/\partial \rho = \Delta \rho_j D_{\rho} f_{i,j+1}; \quad (3.12b)$$

$$a_6 = \partial b(0,1)/\partial x = \Delta \rho_j \partial f(\rho_j, T_{i+1})/\partial \rho = \Delta \rho_j D_{\rho} f_{i+1,j}; \quad (3.12c)$$

$$a_7 = \partial b(1,1)/\partial x = \Delta \rho_j \partial f(\rho_{j+1}, T_{i+1})/\partial \rho = \Delta \rho_j D_{\rho} f_{i+1,j+1}. \quad (3.12d)$$

Similarly,

$$\partial b(x,y)/\partial y = \mathbf{h}'(y)^T \mathbf{A} \mathbf{h}(x), \quad (3.13)$$

and (3.4b) indicates the lower left corner has the  $\partial b/\partial y$ -values:

$$a_8 = \partial b(0,0)/\partial y = \Delta T_i \partial f(\rho_j, T_i)/\partial T = \Delta T_i D_T f_{ij}; \quad (3.14a)$$

$$a_9 = \partial b(1,0)/\partial y = \Delta T_i \partial f(\rho_{j+1}, T_i)/\partial T = \Delta T_i D_T f_{i,j+1}; \quad (3.14b)$$

$$a_{12} = \partial b(0,1)/\partial y = \Delta T_i \partial f(\rho_j, T_{i+1})/\partial T = \Delta T_i D_T f_{i+1,j}; \quad (3.14c)$$

$$a_{13} = \partial b(1,1)/\partial y = \Delta T_i \partial f(\rho_{j+1}, T_{i+1})/\partial T = \Delta T_i D_T f_{i+1,j+1}. \quad (3.14d)$$

As noted above, assuring continuity of first derivatives requires using continuous derivative estimates for the first partial derivatives in (3.12) and (3.14). The derivative estimates used by LEOS are the averages of the two four-point estimates discussed in Section 3.2. In boundary boxes, a non-centered four-point estimate is used to retain cubic precision. As with the standard cubic, this value will be set to zero if it is of the opposite sign from the data slope at the boundary. The use of different derivative estimates means that the bicubic Hermite interpolant to a given set of data does not satisfy exactly the same defining equations as does the standard LEOS bicubic, but the result does have continuous function and first partial derivatives.

The mixed partial derivatives (or “twists”) fill out the remainder of the coefficient matrix. To see this, differentiate either (3.11) or (3.13) with respect to the other variable to obtain:

$$\partial^2 b(x,y)/\partial x \partial y = \mathbf{h}'(y)^T \mathbf{A} \mathbf{h}'(x), \quad (3.15)$$

and the remaining coefficients:

### 3. Coefficient setup

$$a_{10} = \partial^2 b(0,0)/\partial x \partial y = \Delta T_i \Delta p_j \partial^2 f(\rho_j, T_i)/\partial p \partial T ; \quad (3.16a)$$

$$a_{11} = \partial^2 b(1,0)/\partial x \partial y = \Delta T_i \Delta p_j \partial^2 f(\rho_{j+1}, T_i)/\partial p \partial T ; \quad (3.16b)$$

$$a_{14} = \partial^2 b(0,1)/\partial x \partial y = \Delta T_i \Delta p_j \partial^2 f(\rho_j, T_{i+1})/\partial p \partial T ; \quad (3.16c)$$

$$a_{15} = \partial^2 b(1,1)/\partial x \partial y = \Delta T_i \Delta p_j \partial^2 f(\rho_{j+1}, T_{i+1})/\partial p \partial T . \quad (3.16d)$$

Note that the twists could be set to zero without losing the continuity properties, thus reducing the storage requirements to that of the bicubic form, in exchange for a reduction in accuracy. We have not chosen this option for LEOS. Instead, the average of the three-point difference formulas in the two coordinate directions applied to the current first derivative estimates is used to estimate the twists in (3.16).

These formulas are used by the biherm setup routine and result in an interpolant that is exact when interpolating data from a biquadratic function. While higher-order twist estimates could be used to obtain complete bicubic precision, it has been decided that it is not worth the extra effort for such a minimal effect on the interpolant.

The interpolation coefficients are laid out in blocks of 16 in memory, with the coefficients for mesh rectangle  $R_{ij}$  starting at location  $(i*(nr-1)+j)*16$  in the coefficient array.

Note that, since bilinear functions are not a subset of bicubic Hermite functions, there is no convenient way to drop to bilinear inside the two-phase region. Consequently, there is no modification of the biherm interpolant for two-phase data comparable to that discussed in Section 3.3. Experience has shown that this interpolant does “ring” near the phase transition boundary, but the behavior is confined to only boxes adjacent to this boundary and is not nearly as pathological as the standard bicubic. To eliminate this “ringing” altogether, use the bimond interpolant, described in the next section.

#### 3.5. Monotone bicubic Hermite (bimond)

Monotonicity preservation is possible with the Hermite form of the bicubic interpolant (see [4]–[6]). We have extended the algorithm described in [6] to handle piecewise monotonic data (such as a typical pressure table) and included it in LEOS as the bimond option.

The name “bimond” is historical. The original version of our univariate monotone piecewise cubic interpolation algorithm was implemented in subroutine MONDER (MONotone DERivatives), a misnomer for the fact that it determined derivative values that resulted in a monotone piecewise cubic Hermite interpolant. We have retained this name for the univariate routine described in Section 3.10, below. Quite naturally, the bivariate version became known as BIMOND. The incarnation included in the LEOS access library is BIMOND5 (the fifth release).

In brief outline, the BIMOND5 algorithm proceeds as follows:

- Step 1. Initialize two arrays that characterize the monotonicity properties of the data.  
(In each segment where the data have a common monotonicity sense, the

## The LEOS Interpolation Package

interpolant will preserve that monotonicity, except perhaps in boxes adjacent to a switch in data monotonicity).

- Step 2. Compute initial values for the first partial derivatives  $\partial b/\partial x$  and  $\partial b/\partial y$  that satisfy a sufficient condition for monotonicity along the mesh lines. This is done by first initializing these values as described above for the biherm option, and then screening them, possibly reducing derivative magnitudes to satisfy the condition.
- Step 3. Construct intervals of acceptable values, containing zero, for the twists  $\partial^2 b/\partial x \partial y$ . This step may require further reduction in magnitude of first partial derivatives.
- Step 4. Compute values for the twists, as described above for the biherm option, and map them into the intervals determined in Step 3.

The primary complication beyond BIMOND4 [6] occurs in detecting boundaries of monotonicity regions and treating derivative values in adjacent boxes appropriately.

Once the interpolation coefficients have been determined via the BIMOND5 algorithm, the resulting function is evaluated and/or inverted in exactly the same way as an ordinary biherm interpolant. Only the coefficient setup is different.

### 3.6. Biquadratic

For compatibility with the past, the coefficient calculation algorithm used is a C translation of the one in EOS4 [7]. (Further details will not be given here.)

The interpolation coefficients are laid out in blocks of nine in memory, with the coefficients for mesh rectangle  $R_{ij}$  starting at location  $(i*(nr-1)+j)*9$  in the coefficient array.

### 3.7. Linear

A univariate linear function is uniquely determined by its values at two distinct points. The linear interpolant on the mesh interval  $[\rho_j, \rho_{j+1}]$  is thus determined by

$$f_k = f(\rho_k), \quad k = j, j+1. \quad (3.17)$$

Using (2.2) and (2.3), we can write conditions for the coefficients:

$$f_j = f(\rho_j) = l(0) = a_0; \quad (3.18a)$$

$$f_{j+1} = f(\rho_{j+1}) = l(\rho_{j+1}-\rho_j) = a_0 + a_1(\rho_{j+1}-\rho_j). \quad (3.18b)$$

(3.18a) gives us  $a_0$  directly:

$$a_0 = f_j. \quad (3.19a)$$

From (3.18b) and (3.19a) we have

$$a_1 = (f_{j+1} - f_j) / (\rho_{j+1} - \rho_j). \quad (3.19b)$$

### 3. Coefficient setup

#### 3.8. Cubic

A univariate cubic function is uniquely determined by the values of the function and its first derivative at two distinct points. The cubic interpolant on the mesh interval  $[\rho_j, \rho_{j+1}]$  is thus determined by (3.17) and

$$d_k = f'(\rho_k), \quad k = j, j+1. \quad (3.20)$$

Differentiating (2.4) gives

$$f'(\rho) = c'(x) = a_1 + 2a_2x + 3a_3x^2, \quad (3.21)$$

so that we have the four conditions:

$$f_j = f(\rho_j) = c(0) = a_0; \quad (3.22a)$$

$$d_j = f'(\rho_j) = c'(0) = a_1; \quad (3.22b)$$

$$f_{j+1} = f(\rho_{j+1}) = c(\rho_{j+1} - \rho_j) = a_0 + a_1(\rho_{j+1} - \rho_j) + a_2(\rho_{j+1} - \rho_j)^2 + a_3(\rho_{j+1} - \rho_j)^3; \quad (3.22c)$$

$$d_{j+1} = f'(\rho_{j+1}) = c'(\rho_{j+1} - \rho_j) = a_1 + 2a_2(\rho_{j+1} - \rho_j) + 3a_3(\rho_{j+1} - \rho_j)^2. \quad (3.22d)$$

(3.22a) and (3.22b) give us  $a_0$  and  $a_1$  directly:

$$a_0 = f_j; \quad (3.23a)$$

$$a_1 = d_j. \quad (3.23b)$$

Substituting these into (3.22c) and (3.22d) yields a pair of equations to be solved for the remaining two coefficients. From these we obtain:

$$a_2 = -(2\Delta_j + \Delta_{j+1}); \quad (3.23c)$$

$$a_3 = (\Delta_j + \Delta_{j+1}) / (\rho_{j+1} - \rho_j), \quad (3.23d)$$

where

$$\Delta_k = (d_k - m) / (\rho_{j+1} - \rho_j), \quad k = j, j+1,$$

and  $m$  is the data slope,  $m = (f_{j+1} - f_j) / (\rho_{j+1} - \rho_j)$ .

If we use the same derivative estimation scheme to produce  $d_j$  and  $d_{j+1}$  as is used for  $D_\rho f_{mn}$  in the bicubic setup, we obtain a univariate cubic interpolant that is compatible with the bivariate bicubic, in the sense discussed in Section 2.1, above.

#### 3.9. Cubic Hermite

Differentiating (2.5) and using the defining characteristics of the cubic Hermite basis functions (1.7) yields the four formulas:

$$a_0 = c(0) = f(\rho_j) = f_j; \quad (3.24a)$$

$$a_1 = c'(0) = \Delta\rho_j f'(\rho_j) = \Delta\rho_j d_j; \quad (3.24b)$$

$$a_2 = c(1) = f(\rho_{j+1}) = f_{j+1}; \quad (3.24c)$$

$$a_3 = c'(1) = \Delta\rho_j f'(\rho_{j+1}) = \Delta\rho_j d_{j+1}, \quad (3.24d)$$

## The LEOS Interpolation Package

These immediately give us the fact that the cubic Hermite function (2.5) satisfies the conditions (3.17) and (3.20) that are necessary and sufficient to uniquely define a cubic function on  $[\rho_j, \rho_{j+1}]$ .

If we use the same derivative estimation scheme to produce  $d_j$  and  $d_{j+1}$  as is used for  $D_\rho f_{mn}$  in the bicubic Hermite setup, then we obtain a univariate cubic Hermite interpolant that is compatible with its bivariate counterpart, in the sense discussed in Section 2.1, above.

### 3.10. Monotone cubic Hermite (*monder*)

In order to provide a univariate interpolant that is compatible with bimond, in the sense discussed in Section 2.1, we have included an algorithm that uses essentially the procedure of Step 2 of BIMOND5 (see Section 3.5, above) to compute a piecewise monotonic cubic Hermite interpolant. Strictly speaking, *monder* will be compatible with the bimond interpolant only if no Step 3 first derivative modifications were required along the lowest isotherm.

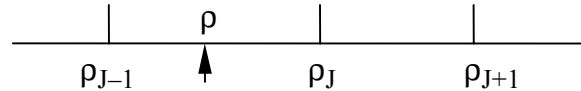
## 4. Forward evaluation

Once we have the array of coefficients  $a_k$  for the desired interpolant to a given set of data, we are in position to evaluate the interpolant at any number of points.

### 4.1. Table lookup

With any functional form, a nontrivial part of the evaluation for a given  $(\rho, T)$  is determining box indices  $i$  and  $j$  such that  $T_i \leq T < T_{i+1}$  and  $\rho_j \leq \rho < \rho_{j+1}$ . The two variables are searched independently. The LEOS interpolation package uses a binary search with guess. The index found at one point is saved and used as a guess at the interval index for the next point. (The index for the first point is initialized to zero.) If the defining conditions are already satisfied by the saved index, we are done. If not, the next interval in the direction of the input value is examined. If that test also fails, then a binary search is performed on the remaining part of the data.

For example, let  $J$  be the saved  $\rho$ -index. If  $\rho_{J-1} \leq \rho < \rho_J$ , the test  $\rho_J \leq \rho$  will fail. From the direction of the failure, the code will then test for  $\rho_{J-1} \leq \rho$ . This succeeds in this case, and the new  $\rho$ -index is  $j=J-1$ . If  $\rho < \rho_{J-1}$ , however, a binary search will then be performed in  $[\rho_0, \rho_{J-1}]$ .



In order to provide thread-safety, the function `leos_lookup` that implements the lookup procedure for a single variable saves no state between calls. It does proceed as indicated, but starts anew for each array it is asked to look up. It returns an array of

## 4. Forward evaluation

indices, which are then passed on to the appropriate evaluation routine for a most efficient computation.

### 4.2. Evaluation

Once the proper mesh rectangle has been located, the variables are transformed according to (1.1) and (1.2) (except for biquadratic), a pointer is set to the appropriate location in the coefficient array, and the appropriate equation is evaluated. If derivatives are requested, the appropriate derivative formulas are also evaluated, and the values are returned to the calling program.

In the bicubic Hermite case, formula (3.10a) can be associated either from the left or right, for two possible nested four-element summations. In one case linear combinations of the x-basis functions are formed, and the results are used to form linear combinations of the y-basis functions. The reverse is the case if the other association is chosen. After some experimentation, different associations have been used in the evaluator, depending on the evaluation history, in an attempt to minimize evaluation time. The result is that a biherm evaluation is only 20 to 40 percent slower than a standard bicubic one (depending on whether derivatives are evaluated).

For bicubic Hermite derivative evaluation, formula (3.11) or (3.13) is used. To provide the necessary values, the Hermite basis function evaluator has an option to return  $\mathbf{h}'$  as well as  $\mathbf{h}$ . The relevant formulas are obtained by differentiating (1.8), namely:

$$h_0'(t) = -6t + 6t^2 = -h_1'(1-t) = -6tu; \quad (4.1a)$$

$$h_1'(t) = 6t - 6t^2 = 6tu; \quad (4.1b)$$

$$h_3'(t) = 1 - 4t + 3t^2 = h_4'(1-t) = u(u - 2t); \quad (4.1c)$$

$$h_4'(t) = -2t + 3t^2 = t(t - 2u). \quad (4.1d)$$

Note that, given these values, one could use (3.15) to evaluate the twist, but this is not a current LEOS option.

### 4.3. Extrapolation

The current LEOS evaluators allow two extrapolation options, controlled by argument `extr_flag`.

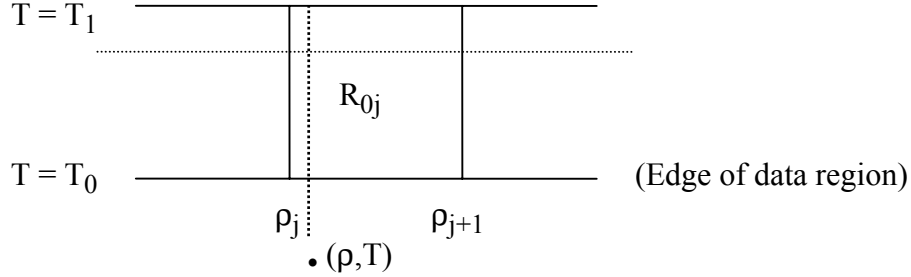
The only option allowed prior to release 7.02 was equivalent to `extr_flag=0`. In this case, if  $(\rho, T)$  is outside the table (i.e.,  $\rho < \rho_0$  or  $\rho > \rho_{nr-1}$ ,  $T < T_0$  or  $T > T_{nt-1}$ ), then the value at the nearest boundary point is returned. (That is, no extrapolation is performed.) If derivatives are requested, zero is returned to match the constant behavior of the extrapolant.

On the other hand, if `extr_flag=1` and  $(\rho, T)$  is outside the table, the value of the function and derivative at the nearest edge point are computed. The linear function determined by these two values is evaluated at  $(\rho, T)$  for the returned value. (This will be linear in the out-of-range variable and the order requested in the other.) The edge

## The LEOS Interpolation Package

derivatives are returned if requested. If both values are out of range, then the bilinear extrapolant determined by the function and derivative values at the nearest corner is used.

As an example, suppose  $\rho_j < \rho < \rho_{j+1}$  but  $T < T_0$ :



In this case, we will have scaled variable values  $0 < x < 1$ , but  $y < 0$ . If `extr_flag=0`, then  $f(\rho, T_0)$ , the value at  $(x, 0)$ , will be returned. If `extr_flag=1`, then  $\partial f(\rho, T_0)/\partial \rho$  will also be computed (even if not requested) and  $f(\rho, T_0) + (T - T_0) \partial f(\rho, T_0)/\partial \rho$  will be returned. NOTE: There is no special test for negative input temperatures or densities in either case.

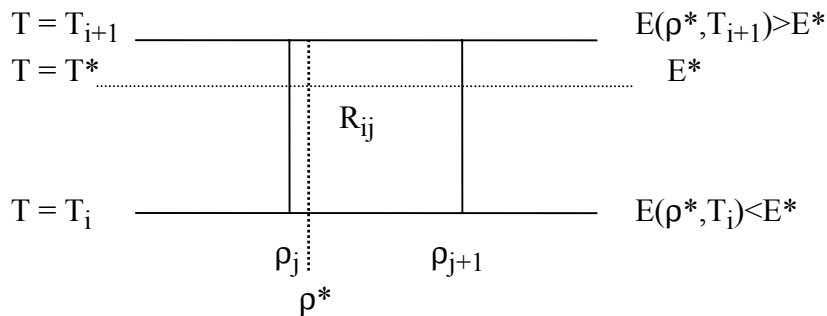
## 5. Inverse evaluation

Many applications use  $\rho$  and  $E$  as the fundamental variables. In order to use the LEOS tables, it is necessary to do an inverse lookup. Given values  $(\rho, E) = (\rho^*, E^*)$ , we need to find a temperature  $T = T^*$  such that

$$E(\rho^*, T^*) = E^*. \quad (5.1)$$

### 5.1. Table lookup

The same lookup procedure described above is used to find  $j$  such that  $\rho_j \leq \rho^* < \rho_{j+1}$ . It is then necessary to examine the values  $E_i = E(\rho^*, T_i)$ ,  $i = 0, 1, \dots, nt-1$  to find an  $i$  such that  $E_i \leq E^* < E_{i+1}$ . A sequential search might require  $nt$  evaluations of  $E$  to determine  $i$ . To reduce code complexity, a simple binary search (without guess) is currently employed. Caution: This assumes that  $E$  is monotonic in  $T$ ; i.e.,  $E_i < E_{i+1}$ ,  $i = 0, \dots, nt-2$ .



We observe that these evaluations are also made simpler by using the fact that  $T = T_i$  implies that  $y = 0$ , except in the biquadratic case.

## 5. Inverse evaluation

### 5.2. Direct inversion (bilinear)

Once the appropriate T-interval has been found, we need to solve equation (5.1) for  $T=T^*$ . In the bilinear case, this is quite simple. From (1.4) we have

$$E(\rho^*, T) = a_0 + a_1 x^* + a_2 y + a_3 x^* y, \quad (5.2)$$

where

$$x^* = (\rho^* - \rho_j) / (\rho_{j+1} - \rho_j).$$

Equation (5.2) can be solved directly for y,

$$y = (E^* - (a_0 + a_1 x^*)) / (a_2 + a_3 x^*), \quad (5.3)$$

and the desired value (obtained by inverting (1.2)) is:

$$T^* = (T_{i+1} - T_i) y + T_i. \quad (5.4)$$

### 5.3. Inverse iteration (bicubic)

The bicubic case is much more complicated. In this case we have to solve a cubic polynomial equation for y. The present code uses a hybrid secant/bisection algorithm to solve this. Matters are simplified a bit by the fact that, due to the variable transformation (1.2), we are solving for a root in the interval [0,1].

The iteration tolerance is currently set at 1.0e-7. (This was 1.0e-5 in an earlier version, but that was deemed insufficient accuracy.) A typical call requires 5–7 iterations, but both smaller and larger values have been observed. (An earlier version that used bisection exclusively required 15 iterations per call.)

### 5.4. Inverse iteration (biherm)

The bicubic Hermite case is handled much like the standard bicubic case. The same iteration procedure is used. The only difference is that the cubic being solved is represented in Hermite form, rather than power form.

### 5.5. Inverse iteration (biquadratic)

The biquadratic case is intermediate in complication. In this case, a quadratic equation has to be solved, once we determine the interval  $[T_i, T_{i+1}]$  containing the target  $T^*$ . For compatibility with the past, the algorithm used is a C translation of the one in EOS4 [7], with some of the special case coding omitted.

### 5.6. Extrapolation

If the input  $\rho$ -value is out of range, the above-mentioned inversion is carried out using the appropriate border strip. If the E-value is out of range, the algorithm returns the associated boundary T-value, with zero T-derivative. (There is no linear extrapolation option.)

## The LEOS Interpolation Package

### 6. The Tcalc option

Because the iterative solution of cubic equations is rather expensive, an inverse evaluation is much more time-consuming than a direct evaluation in the bicubic case. In order to get around this, LEOS provides a Tcalc alternative. This involves setting up a  $(\rho, E)$  mesh and doing an inverse evaluation at each point in the mesh to produce a  $T(\rho, E)$  table. This is done once at setup, and then only forward evaluations are done in this table to produce the T-values required for evaluating other functions during the course of a calculation.

#### 6.1. The Tcalc mesh

Because of the shape of most energy functions, there are no T-values corresponding to small energy values at large  $\rho$ . In order to produce a rectangular mesh for subsequent forward evaluation, it is thus necessary to modify the above idealized procedure to produce a  $T(\rho, \epsilon)$  table, where  $\epsilon = E - E_0$ , and  $E_0 = E_0(\rho) = E(\rho, T_0)$  is the lowest isotherm of the energy table to be inverted.

The Tcalc setup routine in the interpolation package allows several options as to how this  $(\rho, \epsilon)$  mesh is set up. The mesh is derived from the  $(\rho, T)$  mesh for the energy table being inverted. There are two parameters which control the grid points in the  $(\rho, \epsilon)$  mesh. (The word “input” below refers to the energy table mesh.)

$m\epsilon$  is the number of  $\epsilon$ -intervals per input T-interval. The mesh will always have  $\epsilon_0 = 0$ . Normally  $\epsilon_1$  is chosen to be the value of  $E - E_0$  at the left end of the lowest isotherm,  $E(\rho, T_i)$ ,  $i > 0$ , for which  $E - E_0$  is positive.  $\epsilon_{n\epsilon}$  is chosen to be the smallest value of  $E - E_0$  on the highest isotherm,  $E(\rho, T_{nt})$ , where  $n\epsilon = m\epsilon \cdot nt + 1$ . The remaining values will be uniformly logarithmically spaced between them. (There is a special option  $m\epsilon = 0$  which will result in the choice  $\epsilon_i = E(0, T_i) - E(0, T_0)$ , omitting any values which are zero for  $i > 0$ , so that  $n\epsilon \leq nt$ . This option allows one to mimic the spacing of the input T-mesh.)

$m\rho$  is the number of  $\rho$ -intervals per input  $\rho$ -interval. The Tcalc  $\rho$ -mesh will be a refinement of the input  $\rho$ -mesh, generated by inserting  $m\rho - 1$  new logarithmically spaced values in each interval of the original mesh. The resulting mesh will have  $n\rho = m\rho \cdot (nr - 1) + 1$ , and  $m\rho = 1$  gives the original  $\rho$ -mesh.

The current LEOS user-level Tcalc setup routine uses the values  $m\epsilon = 1$  and  $m\rho = 1$ .

#### 6.2. Tcalc setup

Once a  $(\rho, \epsilon)$  mesh has been set up as discussed above, the Tcalc setup routine then does interpolation of the appropriate type in the bottom row of the input energy table to compute  $E_0(\rho) = E(\rho, T_0)$  for each of the  $n\rho$  values in the Tcalc  $\rho$ -mesh. It then cycles over the  $\epsilon$ -mesh, computes  $E = \epsilon + E_0(\rho)$  at each point, and calls the appropriate inverse evaluation routine to compute the Tcalc table,  $T(\rho, \epsilon)$ . It then sets up coefficients for interpolation in this table, based on the interpolation method used for the original E-table.

## 7. Package organization

All of the information needed for interpolation in the Tcalc table is contained in the mesh and the interpolation coefficients, so the user never actually sees the  $T(\rho, \epsilon)$  array.

For completeness, the user is allowed to use the Tcalc option for a bilinear energy table, but this is really not a good idea, because the direct inversion of a bilinear function is much faster. There is no biquadratic Tcalc option.

### 6.3. Tcalc evaluation

Because it is necessary to do a one-dimensional interpolation in the bottom row of the original energy table to compute the  $E_0(\rho)$  needed to convert a user's E-value into the associated  $\epsilon$ -value for interpolation in the Tcalc table, there is a special evaluator for this purpose. The resulting interpolation, while somewhat more expensive than a normal bicubic or biherm evaluation, is much faster than an inverse bicubic or biherm evaluation.

### 6.4. Tcalc extrapolation

The same extrapolation options described in Section 4.3 are also provided for the Tcalc evaluator. If the  $\rho$ -value is out of range, the appropriate extrapolation is first performed when computing  $E_0(\rho)$ . The resulting  $(\rho, \epsilon)$ -values are then passed to the appropriate direct evaluator, which performs any necessary additional extrapolation.

## 7. Package organization

The LEOS interpolation package is organized in three sub-packages. There are one- and two-dimensional sub-packages, `interp1D` and `interp2D`, and a set of utility routines that are used by both of these. The first two require a header file, such as `LEOS_proto.h`, which defines the interpolation type parameters `LEOS_BILINEAR`, etc., and all require the header file `LEOS_Ftype.h`, which defines the Fortran-compatible types used throughout LEOS. In order to support the new biherm interpolator and new extrapolation options introduced since [2], some routines have been added and some function names changed.

### 7.1. The interpolation utilities

The LEOS interpolation utilities are contained in files `leos_int_util.c`, `leos_lookup.c`, and `leos_intrp_vers.c`, with header file `LEOS_int_util.h`. (`leos_lookup` is packaged separately, because it may have uses independent of LEOS, as is `leos_intrp_vers`.) There is also a file `pchsubs.c`, with header file `LEOS_PCH.h`, which is used by the `bimond` option. This contains C versions of routines from the REAL\*8 version of the univariate PCHIP package.

The contents of these files are as follows:

## The LEOS Interpolation Package

`leos_int_util.c`:

- `leos_dcopy`: copy a double precision array.
- `leos_dswap`: swap two double precision arrays.
- `leos_p2d`: compute a quadratic polynomial and evaluate its derivative for cubic setup routines.
- `leos_p3d`: compute a cubic polynomial and evaluate its derivative for cubic setup routines.
- `leos_cubic_derivs`:  
compute derivative estimates suitable for cubic Hermite interpolation.
- `leos_hbasis`: evaluate univariate Hermite basis functions.
- `leos_fun3c`: compute a 3-point derivative approximation.
- `leos_mach`: returns Real8 (double) floating point constants.
- `leos_fsign`: routine to emulate the Fortran SIGN function.
- `leos_sign_test`:  
modified PCHIP8 sign-testing routine.

`leos_lookup.c`:

- `leos_lookup`: look up an array in an ordered 1-D table.

`leos_intrp_vers.c`:

- `leos_intrp_vers`: return the version number for the package.

`pchsubs.c` (C versions of PCHIC8 subsidiary routines):

- `pchcs8_`: adjusts the values of derivatives in the vicinity of a switch in direction of data monotonicity for a "visually pleasing" curve.
- `pchsw8_`: PCHCS8 switch excursion limiter.

### **7.2. Interp1D**

The one-dimensional interpolation routines are contained in the `interp1D` sub-package. The source files are `leos_int1D_setup.c` and `leos_int1D_eval.c`, with header file `LEOS_int1D.h`.

The contents of these files are as follows:

`leos_int1D_setup.c` (coefficient set-up functions):

- `leos_setup_linear`: set coefficient arrays for linear interpolation.
- `leos_setup_cubic`: set coefficient arrays for cubic interpolation.
- `leos_setup_hermit`: set coefficient arrays for cubic Hermite interpolation.
- `leos_setup_monder`: set coefficient arrays for monotone cubic Hermite (monder) interpolation.

`leos_int1D_eval.c` (direct interpolation functions):

## 7. Package organization

`leos_evalu_univar`: evaluate interpolant at an array of points. This function gets the index array from `leos_lookup` and then calls the appropriate one of the following functions.

`leos_evalu_linear`: implements linear interpolation option.

`leos_evalu_cubic`: implements cubic interpolation option.

`leos_evalu_hermit`: implements cubic Hermite interpolation option.

### 7.3. Interp2D

Most of the interpolation package is contained in the `interp2D` sub-package. The setup routines and the direct and inverse evaluation routines are in files `leos_int2D_setup.c`, `leos_int2D_eval.c`, and `leos_int2D_inv.c`, with header file `LEOS_int2D.h`. For packaging purposes, the bimond option is implemented in separate files `leos_setup_bimond.c` and `pbhpm.c`, which require the new header file `LEOS_PBH.h` in addition to `LEOS_int2D.h`.

The contents of these files are as follows:

`leos_int2D_setup.c` (coefficient set-up functions):

`leos_setup_bilinear`: set coefficient arrays for bilinear interpolation.

`leos_setup_biquad`: set coefficient arrays for biquadratic interpolation.

`leos_setup_bicubic`: set coefficient arrays for bicubic interpolation.

`leos_setup_bicubic2p`: set coefficient arrays for bicubic interpolation, with modifications for two-phase data.

`leos_bicubic_matrix`: set up and factor interpolation matrix (3.6) for `leos_setup_bicubic` or `leos_setup_bicubic2p`.

`leos_setup_biherm`: set coefficient arrays for biherm interpolation.

The following two general-purpose utility functions are used only by the bicubic setup routines, so are included in `leos_int2D_setup.c`:

`leos_factor`: compute the LU factorization of a matrix.

`leos_solve`: solve a system, given its LU factorization.

`leos_setup_bimond.c` (coefficient set-up for bimond):

`leos_setup_bimond`: set coefficient arrays for bimond interpolation. This merely allocates temporary storage and calls `PBHpm`.

`pbhpm.c` (BIMOND5 set-up algorithm):

`PBHpm`: main control routine for the BIMOND5 algorithm (see Section 3.5). This is the result of converting Fortran subroutine `PBHpm` (Piecewise Bicubic Hermite interpolation which preserves Piecewise Monotonicity) and its 14 subsidiaries to C and significantly cleaning up the result.

`pbhinit_`: initialize global variables for `PBHpm`.

## The LEOS Interpolation Package

pbhm1a\_: implements Step1 of the BIMOND5 algorithm.

pbhm2b\_: implements Step2 of the BIMOND5 algorithm.

pbhm3a\_: implements Step3 of the BIMOND5 algorithm.

pbhm4a\_: implements Step4 of the BIMOND5 algorithm.

(Lower-level subsidiaries are not included in this list. Refer to the comments at the beginning of PBHpm if you really want to know the code structure.)

leos\_int2D\_eval.c (direct evaluation functions):

leos\_evalu\_bivar: evaluate interpolant at an array of (T,p) points. This function gets the index arrays from leos\_lookup and then calls the appropriate one of the following functions.

leos\_evalu\_bilinear: implements the bilinear interpolation option.

leos\_evalu\_biquad: implements the biquadratic interpolation option.

leos\_evalu\_bicubic: implements the bicubic interpolation option.

leos\_evalu\_biherm: implements the bicubic Hermite interpolation option.

leos\_int2D\_inv.c (inverse evaluation functions):

leos\_inverse\_vals: find T's corresponding to array of (p,E). This function gets the p-index array from leos\_lookup and then calls the appropriate one of the following functions.

leos\_inv\_bilinear: implements bilinear inverse interpolation.

leos\_inv\_biquad: implements biquadratic inverse interpolation.

leos\_inv\_bicubic: implements inverse interpolation for the “standard” bicubic form.

leos\_inv\_biherm: implements inverse interpolation for the bicubic Hermite form.

The routines that handle the Tcalc option are packaged separately. They are contained in file leos\_int2D\_tcalc.c, with header file LEOS\_tcalc.h.

The contents of this file are as follows:

leos\_setup\_tcalc: set up ( $\epsilon$ ,p) mesh, compute T on this mesh, and return coefficients for interpolating in this Tcalc table.

leos\_evalu\_tcalc: interpolate in a previously created Tcalc table.

## 8. Possible enhancements

# 8. Possible enhancements

### **8.1. *Lookup improvements***

Since the tabulation points are logarithmically spaced, it may be possible to further speed up the lookup phase by introducing a hash table or other device. It will be necessary to do this without incurring the expense of a logarithm or exponential call during evaluation.

### **8.2. *Newton iteration***

Since derivative of a cubic is relatively inexpensive to evaluate, it might be possible to further speed up the bicubic inverse iteration by changing from the secant method to a safeguarded Newton iteration. Since the iteration is to be done on  $[0,1]$ , it should be easy to tell when we are heading out of the interval and take appropriate corrective action.

# The LEOS Interpolation Package

## References

- [1] Fritsch, Frederick N., The LEOS Bivariate Interpolation Package, Unpublished report (10 June 1998).
- [2] Fritsch, Frederick N., The LEOS Interpolation Package, Unpublished report (15 November 2001).
- [3] Fritsch, Frederick N., The LEOS Interpolation Package, Second Edition, UCRL-ID-148544 (May 21, 2002).
- [4] Carlson, R.E., and F.N. Fritsch, “Monotone piecewise bicubic interpolation”, SIAM J. Numer. Anal., Vol. 22, No. 2 (April 1985), pp.386–400.
- [5] Carlson, R.E., and F.N. Fritsch, “An algorithm for monotone piecewise bicubic interpolation”, SIAM J. Numer. Anal., Vol. 26, No. 1 (February 1989), pp.230–238. [The associated Fortran code, BIMOND3, was documented in UCID-21143 (August 1987).]
- [6] Carlson, R.E., and F.N. Fritsch, “A bivariate interpolation algorithm for data which are monotone in one variable”, SIAM J. Sci. Stat. Comput., Vol. 12, No. 4 (July 1991), pp.859–866. [The associated Fortran code, BIMOND4, was never formally documented.]
- [7] Chase, Lila, EOS4 User Manual, Internal Report UCIR-1436a, Rev. 33 (July 7, 1999).

## Acknowledgements

I wish to thank David Young for supporting this project and for his patience with me while I tried to “get it right”. Thanks are due also to Ellen Hill for helping integrate this package into the LEOS access library and to several LEOS users for their suggestions.

This work was performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48.